NHL Season Goals and Playoff Prediction

Data Science 4: Machine Learning

Group 8 Project Report:

MacDougald, Michael
Punja, Nidhi
Siddiqi, Areeb
Tang, Chen
YAN, Xin Wei
Yi, Seo

August 2nd 2021

Table of Contents

Objective:

The National Hockey League (NHL) is a men's professional ice hockey league in North America and is comprised of 32 teams (7 in Canada and 25 in the United States). NHL was established in Montréal, Québec, in 1917 and is now considered to be the world's premier professional ice hockey league. It is also one of the major professional sports leagues in the United States and Canada.

The NHL season is divided into a preseason (September- early October), a regular season (from early October - early to mid-April) and a postseason, the Stanley Cup playoffs (mid-April – early June). The Stanley Cup is the most prestigious team award and is awarded annually to the league playoff champion at the end of each season.

Group 8 has decided to analyze and explore the NHL Game and Player data with the objective to see if -

1. Season goals scored by players could be predicted
2. There are factors that can contribute to a team being qualified for the playoffs (Stanley Cup)

Data Preparation:

Data Source:

The NHL recently made all their statistics publicly available from the last 100 years of league play. This includes various statistics for teams, individual players, and every game recorded. This massive amount of data opens all kinds of possibilities for analysis. While it is officially available on the NHL API, there is no official documentation, and any information on the statistics and tables available is mostly from trial and error on forums and community sites.

1) Playoff Data:
   We pulled 11 years of playoff information from these APIs to determine which teams made the playoffs using the links below
   https://records.nhl.com/site/api/playoff-series
   https://statsapi.web.nhl.com/api/v1/game/{}/linescore -
   Note - the above link works when we enter in a game ID

2) Seasons Data: We pulled 11 years (seasons) of data using the most recent years played. The data was retrieved by a combination of the links below:
   a. https://records.nhl.com/site/api/franchise
   b. https://records.nhl.com/site/api/franchise-season-results?cayenneExp=franchiseId={}&sort=seasonId&dir=DESC
   c. https://statsapi.web.nhl.com/api/v1/teams/{}?expand=team.roster&season={}
   d. https://statsapi.web.nhl.com/api/v1/people/{}/stats?stats=statsSingleSeason&season={}

We first get the list of franchise (a) and then for each franchise id we got the seasonal result sorted by season id (b). We then used the franchise and season id to get the roster for each team/season(c) and finally used the playerid and season id to get player statistics for that season.

3) Equipment and Salary Data:
We pulled additional player salary and equipment brand/type information from the link below
NHL 2017-18 - Hockey Abstract
Base player information was obtained here. This included demographic and some personal information like height, weight, hometown etc.
https://statsapi.web.nhl.com/api/v1/people/{}
The above link works when we put in an NHL ID (see example for 8470595) -
https://statsapi.web.nhl.com/api/v1/people/8470595

The NHL data contains unique IDs for each team/franchise and unique player IDs which made joining the data a fairly easy task since we could merge based on those matching IDs. We used the base player information as the starting point, and joined in each player's statistics for every year played, added their salary and equipment information, and the information on whether their team made the playoffs. The playoff data was grouped to provide a boolean result for each team on whether they had made the playoffs in each noted season before being joined.

Data Cleaning:

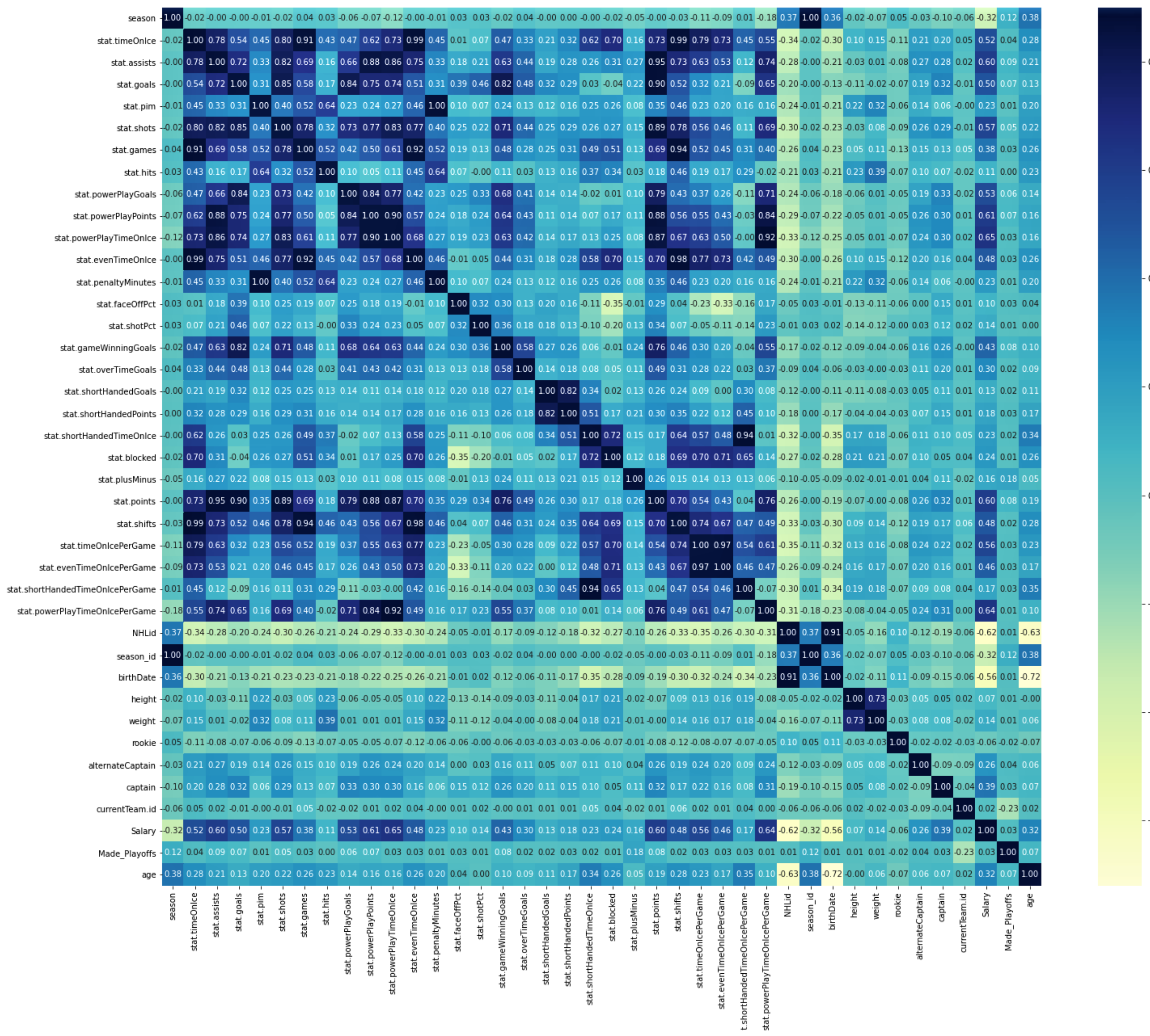
Being from an official source, the data was clean from the start and required only minor cleanup and changes. These are the main changes that were made to the dataset:

1) The majority of the cleanup was filling some NaN values or changing them to an 'other' category.
2) We had to remove players who were noted as active on the roster (under contract) but playing in a minor league since they wouldn't have a full season of data, and would skew the results.
3) We removed goalies from the dataset so that we had only scoring skaters
4) Time related statistics had to be changed from an object data type to numerical values:
    a) timeOnIce
    b) powerPlayTimeOnIce
    c) evenTimeOnIce
    d) shortHandedTimeOnIce
    e) timeOnIcePerGame
    f) evenTimeOnIcePerGame
    g) shortHandedTimeOnIcePerGame
    h) powerPlayTimeOnIcePerGame
5) Changed height of players from feet and inches to centimeters
6) Created an age column for players using season year and birth year for the player
7) Transformed rookie from a boolean True = 1, False = 0

Our final dataframes for the machine learning component consisted of a training/test set with 10 seasons from 2010-2011 to 2019-2020 (TSData), and a holdout set (holdoutSeason) for the 2020-2021 season for final testing.

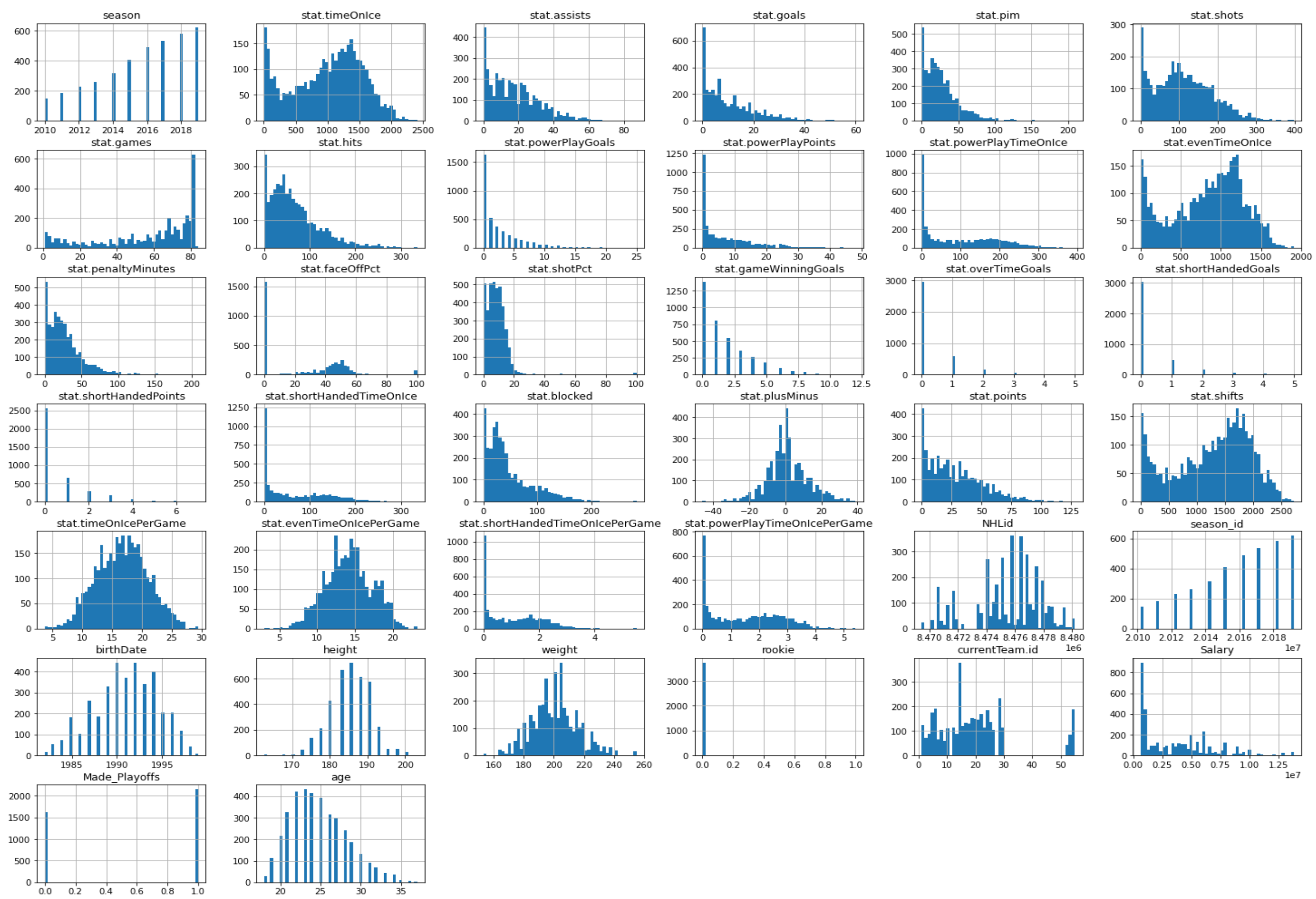
Exploratory Data Analysis:

Correlation Matrix:



The final dataframe for the analysis had 53 columns and 3655 rows. Of the 53 columns, 27 columns represented various statistics for each player (like power play goals, time on ice, penalty minutes etc), 9 were demographic information for each player and the rest were associated with equipment and positions played by each player. We ran a correlation matrix to check for multicollinearity and found that a lot of the statistics fields were collinear with each other, for example stat.point collinear with stats.assist and stat.shots. Salary was also correlated with some of these statistics which could also skew the data. But there is some random collinearity with NHL id and birthdate which isn't helpful or meaningful. So it made sense to remove some of the highly correlated data from the Machine Learning algorithm. We also excluded some statistics percent columns (fields ending in PCT) which were based on the statistics provided for each player. (larger image is available in the notebook)

Histogram:



The data is skewed with the exception for player demographic data (height, weight, birthdate) and time on ice spent by players.

Based on the exploratory data analysis we reduced the number of features from 53 to 16 features (8 categorical columns and 8 numerical columns) to analyze the goals scored by player data.

For the Playoff data, we consolidated the data to show one row per team, per year by taking the mean of demographic data and taking the sum of the statistics data.

Model Design:

Goals Scored by Players:

Because of the multicollinearity within the dataset we used mostly ensemble tree and boosting methods for our analysis. We ran the 4 models below to see if we can predict the number of goals scored by player-

1) Random Forest Regressor without Dimensionality reduction
2) Random Forest Regressor with Dimensionality reduction
3) Gradient Boosting Regressor with GridSearchCV
4) AdaBoost Regressor with RandomizedSearchCV

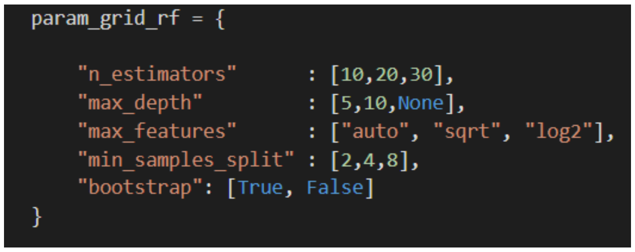
There was some preprocessing of the data done prior to feeding it to all the models:

1) Split the training data to 80% training and 20% testing
2) One Hot Encoded (OHE) some of the categorical values
3) Scaled the numerical data using Standard Scaler

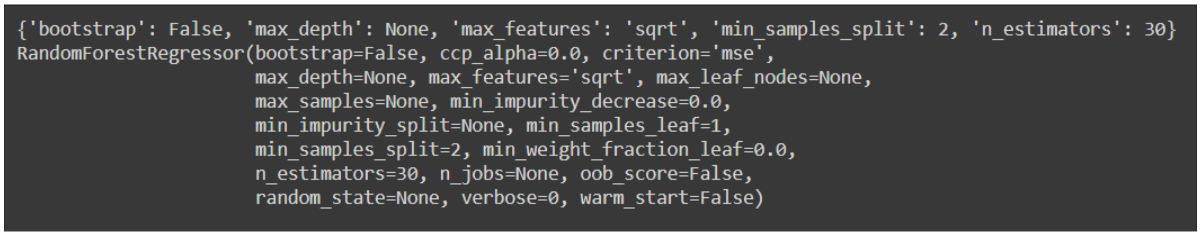
Random Forest Regressor without Dimensionality reduction

We chose to run the RandomForestRegressor model because it is a fast and robust model. The model uses multiple DecisionTrees and at each step it randomly selects a subset of features instead of searching over all the features for each best splitting feature and threshold. The algorithm results in greater tree diversity, which trades a higher bias for a lower variance, generally yielding an overall better model and it avoids overfitting.

We then used GridSearchCV to figure out the best hyperparameters for the model. The inputs given for the hyperparameter search were:


```
param_grid_rf = {

    "n_estimators"      : [10,20,30],
    "max_depth"         : [5,10,None],
    "max_features"      : ["auto", "sqrt", "log2"],
    "min_samples_split" : [2,4,8],
    "bootstrap": [True, False]
}
```

While selecting the inputs, we wanted to try several possible combinations for the analysis and the grid search resulted with the hyperparameters below:

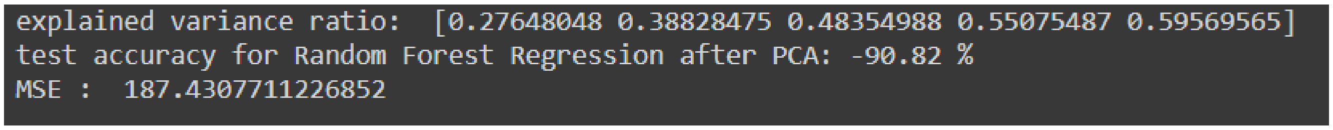


```
{'bootstrap': False, 'max_depth': None, 'max_features': 'sqrt', 'min_samples_split': 2, 'n_estimators': 30}
RandomForestRegressor(bootstrap=False, ccp_alpha=0.0, criterion='mse',
                      max_depth=None, max_features='sqrt', max_leaf_nodes=None,
                      max_samples=None, min_impurity_decrease=0.0,
                      min_impurity_split=None, min_samples_leaf=1,
                      min_samples_split=2, min_weight_fraction_leaf=0.0,
                      n_estimators=30, n_jobs=None, oob_score=False,
                      random_state=None, verbose=0, warm_start=False)
```

The model actually did really well with an R-squared score of 0.91 and a mean squared error value of 8.57

Random Forest Regressor with Dimensionality reduction

Since we had over 50 features in our analysis with multicollinearity issues, we also decided to use the dimensionality reduction technique of Primary Component Analysis (PCA) on the dataset (included all 53 features here and not the limited 23 features) and rerun the RandomForest Regressor. We reduced the dataset to show 5 components, fit the model to the reduced dataset and then tested on the test data. We were surprised to see that the data the R-squared score was negative at -0.90 and the mean squared error value increased to 187.43. We tried playing with the n_components entering higher values (to increase the cumulative variance to 0.95) but it did not improve the score. Suffice to say that this model clearly is not a great fit for this dataset.



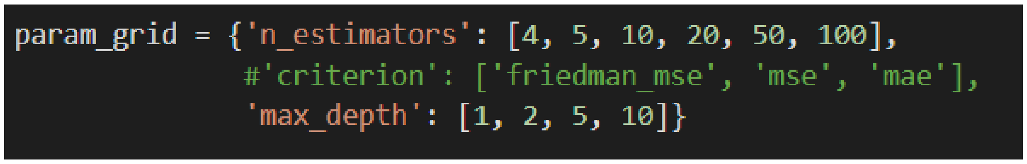
```
explained variance ratio:  [0.27648048 0.38828475 0.48354988 0.55075487 0.59569565]
test accuracy for Random Forest Regression after PCA: -90.82 %
MSE :  187.4307711226852
```

Because the Random Forest Regressor does well on it’s own without dimensionality reduction, we won’t be testing this model on the holdoutseason data (our test dataset)

Gradient Boosting Regressor with GridSearchCV

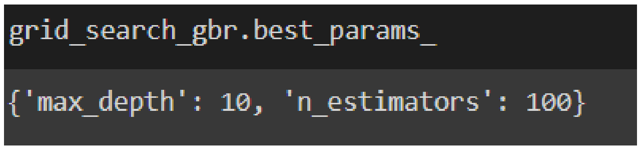
GradientBoosting Regressor is one of the boosting algorithms that combine several weak learners (Decision Trees) into a strong one and train predictors sequentially, each trying to correct it's predecessors. Gradient Boosting Regressor tries to fit a new predictor to the residual errors made by the previous predictor instead of modifying the weight of the training instances at each step. So we picked the Gradient Boosting Regressor algorithm as it is more robust to outliers.

We again used GridSearchCV to figure out the best hyperparameters for the model. The inputs given for the hyperparameter search were:



```
param_grid = {'n_estimators': [4, 5, 10, 20, 50, 100],
              #'criterion': ['friedman_mse', 'mse', 'mae'],
              'max_depth': [1, 2, 5, 10]}
```

As with Random forest, n_estimators represent the number of trees and the max_depth represents the depth limit for each tree. This combination of hyperparameters would give us a varied number of trees for our analysis. The grid search yielded the results below:



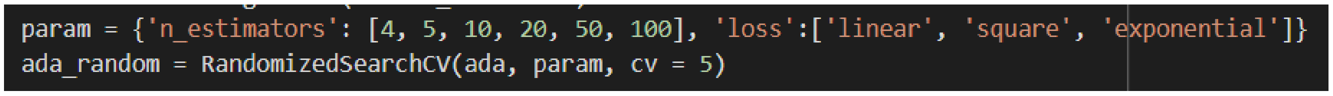
```
grid_search_gbr.best_params_

{'max_depth': 10, 'n_estimators': 100}
```

When we run the model again using these hyperparameters, we get a R-squared score of 0.9 and a mean squared error value of 9.87

AdaBoost Regressor with RandomizedSearchCV

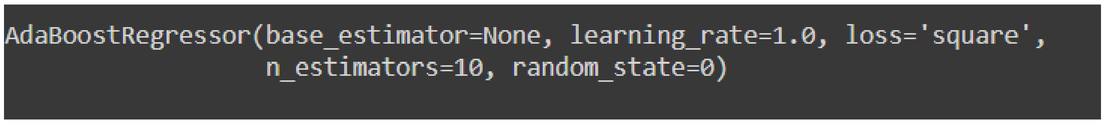
AdaBoost Regressor is another boosting algorithm which uses Decision Trees as a base model. AdaBoost works by putting more weight on difficult to classify instances and less on those already handled well.

We used RandomizedSearchCV to look for the best hyperparameters for the model. The inputs given for the hyperparameter search were:



```
param = {'n_estimators': [4, 5, 10, 20, 50, 100], 'loss':['linear', 'square', 'exponential']}
ada_random = RandomizedSearchCV(ada, param, cv = 5)
```

With randomizedsearchCV not every combination of hyperparameters is tried. We used this mainly because we wanted to just test our data with the AdaBoost Regressor algorithm. We have 2 models that have done fairly well but wanted to check it against another more to see if we could have improved the R-squared score and the mean square error value. The search yielded the parameters below for the model:



```
AdaBoostRegressor(base_estimator=None, learning_rate=1.0, loss='square',
                  n_estimators=10, random_state=0)
```

When we test this model on the test data we get a R-squared score of 0.73 and a mean squared error value of 26.63

Playoff Data:

Looking at consolidated team data, we tried to classify each team as making the playoffs or not in a given season.

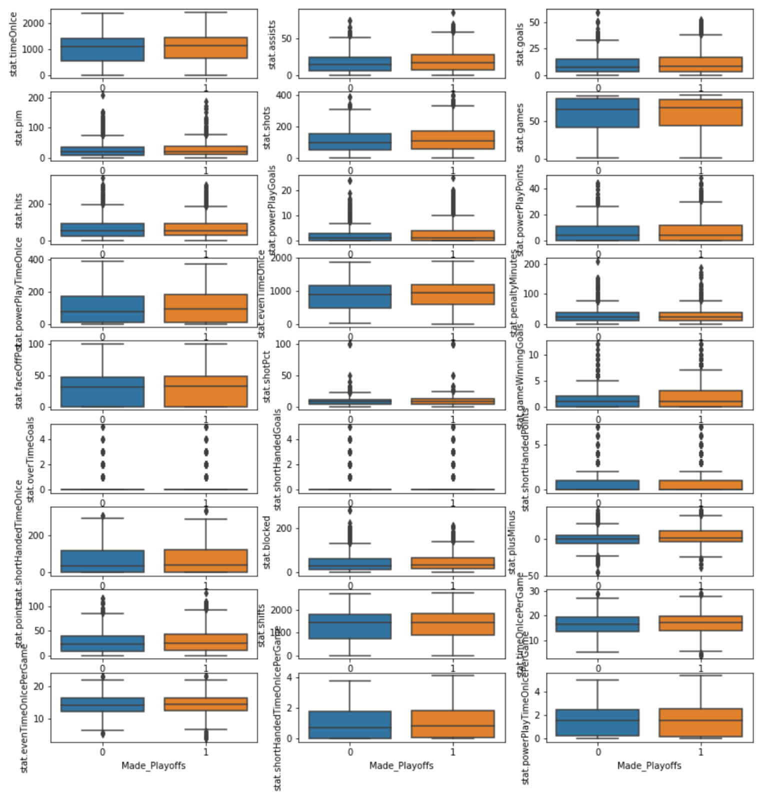
We ran the classifiers below to predict whether a team made it to the playoffs or not:

1) Random Forest Classifier
2) AdaBoost Classifier
3) Extra Trees Classifier
4) Gradient Boosting Classifier

We ran all the above classifiers using the default hyperparameters and the accuracy scores are:

5) Random Forest Classifier: 57.14%
6) AdaBoost Classifier: 63.49%
7) Extra Trees Classifier: 52.38 %
8) Gradient Boosting Classifier: 63.49%

Based on the features used, we didn't see much difference in the mean of teams who did or did not make the playoffs:



Looking at the playoffs a different way, and trying to predict based on individual player data, we were able to get slightly better results in the main train/test data for 10 seasons at 68.8% accuracy without tuning the model, and the same results (~68%) when tested on the holdout dataset for 2020-2021. This data set was slightly different from the last set used for the above 4 algorithms and included OHE categorical columns as well which could explain some of the increased accuracy. After hyperparameter tuning, we had far better model performance on the training data, with a prediction of whether an individual player's team would make the playoffs in a given season of up to 82%. However, regardless of how we tuned the model it was overfitting and we were only able to achieve a best of 68% accuracy on the holdout data, with poorer results in the low 60's as we achieved higher scores on the training data.

Model Evaluation:

We set aside 2020-2021 season data as our test set. Below is the performance of 3 of the 4 models for predicting goals scored by a player:

Random Forest Regressor without Dimensionality reduction: We got an R-squared value of **0.75**

and a mean-squared error of **12.15**. So this model can be used to help predict the scores earned by a hockey player by game and season.

Gradient Boosting Regressor with GridSearchCV: R-squared value of **0.7** and a mean squared error value of **14.71**

AdaBoost Regressor with RandomizedSearchCV:. We got an R-squared value of **0.63** and a mean squared error value of **18.04**

Conclusions:

Our goal was to predict:

1. Season goals scored by players
2. Factors that can contribute to a team being qualified for the playoffs (Stanley Cup)

To do this we ran multiple regressor and classifier algorithms. We found that we could predict the season goals scored by players with a relatively high accuracy, however our models for predicting the playoff data gave suboptimal results where the highest accuracy score on the holdout data was 68%.

Random Forest Regressor without Dimensionality reduction: This model performed the best with the overall R-squared value of 0.91 and a mean squared error value of 8.57. When we tested this model on the holdoutseason data of 2020-2021, we got an R-squared value of **0.75** and a mean-squared error of **12.15**. So the model did not generalize as well when predicting the goals scored by a hockey player.

Random Forest Regressor with Dimensionality reduction: This model performed the worst and we received a negative R-squared value of -0.90 and a mean squared error value of 187.43. Suffice it to say we did not see the point of testing this model on the holdseason data of 2020-2021

Gradient Boosting Regressor with GridSearchCV: This was the second best performing model with the R-squared value of 0.9 and a mean squared error value of 9.87

AdaBoost Regressor with RandomizedSearchCV: This model did not perform as well. We got an R-squared value of 0.75 and a mean squared error value of 26.63

The **Random Forest Regressor without Dimensionality reduction** model can be improved by adding more years worth of data or playing around more with the features picked for the analysis. Since we were using 10 years worth of data, we found that some of the features in more recent data were not always available in the older data sets. So in order to maintain consistency, we only picked features that were the same across all years (this does not apply to the player demographic data). So more data and additional columns might help improve the model created.

References:

1. Drew Hynes, D. H. (n.d.). Drew Hynes / nhlapi. GitLab. Retrieved July 11, 2021, from https://gitlab.com/dword4/nhlapi
2. Vollman, R. V. (2018, April 19). NHL 2017–18 - Hockey Abstract [Dataset]. http://www.hockeyabstract.com/testimonials/nhl2017-18